REMARKS

Claims 1-33 are pending. Claims 1, 4, 10, 13, 22, and 28-33 are amended herein.

The Examiner rejected claims 10-18 and 33 under 35 U.S.C. § 101 as reciting software per se. Although applicant respectfully disagrees, applicant has nevertheless amended these claims to include hardware that defines structural and functional interrelationships between the other claimed elements that permit the functionality of the elements to be realized. Accordingly, applicant respectfully requests that this rejection be withdrawn.

The Examiner rejected claims 4, 13, 28-30, and 32 under 35 U.S.C. § 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention. Applicant has amended these claims to address the Examiner's concerns. Accordingly, applicant respectfully requests that this rejection be withdrawn.

The Examiner rejected claims 1-5, 8-14, and 17-33 under 35 U.S.C. § 103(a) over Nilsson (U.S. Patent No. 6,289,446), and claims 6-7 and 15-16 under 35 U.S.C. § 103(a) over Nilsson in view of Slingwine (U.S. Patent No. 6,219,690). Applicant respectfully traverses this rejection.

Nilsson describes a typical usage of the Unix setjmp and longjmp functions. The Unix setjmp and longjmp functions are used to control the flow of execution of a thread. A thread calls setjmp at a location in a program where execution should return if longjmp is ever called by the same thread. When longjmp is called, the thread returns to the setjmp location even if the thread is several layers deep in a call stack of other functions. Following is a simple example in C++ of using setjmp and longjmp:

```
main()
{
    jmp_buf env;
    int val;
```

```
    val=setjmp(env);
    printf ("val is %d\n",val);
    if (!val) longjmp(env, 1);
    return 0;
}

Output:
val is 0
val is 1
```

When setjmp is called normally, it returns zero, so on the first pass the printf statement indicates that "val is 0." When longjmp is called, execution of the thread passes to the setjmp location and setjmp returns the value of the second parameter passed to longjmp, which in this case is one. Thus, the second time printf is called it indicates that "val is 1." Nilsson describes the traditional usage of setjmp and longjmp, in which these functions were always called from the same thread. One common Unix reference states, "[t]he longjmp() function restores the environment saved by the most recent invocation of setjmp() in the same thread." *The Single Unix Specification, Version 3*, January 31, 2002 (emphasis added).

In contrast, applicant's technology describes a new implementation of setjmp and longjmp that allows inter-thread long jumps between two threads. Although this technique may be used in other architectures, applicant uses this technique in relation to applicant's multithreaded architecture (MTA) processor. The MTA processor has 128 streams that can each be simultaneously executing a thread. Specification, ¶4. A stream can be thought of as a virtual processor. The MTA processor is able to simultaneously execute many tasks, each using a different stream or set of streams. This gives rise to various new signaling techniques for which applicant developed the inter-thread long jump that is the subject of applicant's claims.

Each of applicant's claims describes one thread that calls a set jump function to identify a set jump location, and a different thread that either calls a long jump function directly or causes the first thread to call the long jump function. Claims 1-9 recite "the set

jump location set by a set jump thread" and "when the set jump thread is not the same thread that is currently executing, setting a state of the set jump thread to execute a long jump indicating the set jump location." Claims 10-18 recite "a component executing on the processing component that, when the set jump thread is not the same thread that is currently executing, sets a state of the set jump thread to transfer control to the set jump location." Claims 19-21 recite "under control of a set jump function, storing a current stream state, the current stream state including a return address" and "under control of a long jump function, when the long jump function is invoked by a thread that is different from a thread that invoked the set jump function . . . setting a program counter in the located state information to point to an instruction that invokes the long jump function." Claims 22-32 recite "in a second thread different from a set jump thread . . . causing, based on the determined state, the set jump thread to execute at a set jump location." Claim 33 recites "under control of the long jump thread, causing the set jump thread to jump to the set jump location."

Neither Nilsson nor Slingwine describes an inter-thread long jump. As discussed above, the disclosure of long jumps in Nilsson relates to a traditional long jump in the same thread that called setjmp. The Examiner asserts that it would have been obvious to "set the state of the set jump thread to execute a long jump if the set jump thread is not the same thread that is currently executing." Office Action, pp. 5-6. Applicant respectfully disagrees. In previous architectures, using setjmp and longjmp to jump between threads would be contrary to the intended purpose. The purpose of a long jump in a traditional architecture is, when a particular thread hits an error, to return the thread to a safe point of execution under its own control. There is nothing in Nilsson to suggest that any other thread be involved in the process, or that a second thread be used to cause a long jump in a first thread. Indeed, it is only applicant's specification that describes an inter-thread long jump.

In addition, the Examiner suggests that "[o]ne would be motivated by the desire to ensure that the long jump is executed since the function call has been made." Office

Action, p. 5. The Examiner's suggested motivation is not a motivation for an inter-thread long jump, but rather a motivation for executing the intra-thread long jump that is provided by Nilsson. It is not clear if "the function call" the Examiner is referring to is the setjmp call or the longjmp call. Taking each separately, we first assume that the Examiner is referring to the longjmp call. If the first thread has already made the longjmp call, then there would be no purpose to having a second thread to call it also. We assume then that the Examiner is referring to the setjmp call. If the first thread has already made the setjmp call, then one would be motivated to execute an intra-thread long jump within the first thread. Since Nilsson describes no inter-thread long jump, one would not be motivated to use such a long jump just because a set jump call was made.

Thus, applicant's claims recite elements that are not taught or suggested by Nilsson or Slingwine. Accordingly, applicant respectfully requests that this rejection be withdrawn. If the Examiner maintains the rejection, then applicant respectfully requests that the Examiner provide a reference for the concept of a second thread causing a first thread to execute a long jump.

Based upon these remarks and amendments, applicant respectfully requests reconsideration of this application and its early allowance. If the Examiner has any questions or believes a telephone conference would expedite prosecution of this application, the Examiner is encouraged to call the undersigned at (206) 359-3265.

If any additional fees are due, please charge our Deposit Account No. 50-0665, under Order No. 324758001US4 from which the undersigned is authorized to draw.

Dated: December 13, 2007

Respectfully submitted,

By _____
J. Mason Boswell
Registration No.: 58,388
PERKINS COIE LLP
P.O. Box 1247
Seattle, Washington 98111-1247
(206) 359-8000
(206) 359-7198 (Fax)
Attorney for Applicant